Introduction to STAN

A probabilistic programming language

Songpeng Zu

14 October 2020

Content[1]

An introduction to STAN, a well-designed and easily used tool, for statistical modeling.

- What is STAN.
- How STAN works.
- How to write a STAN script.
- How to run it and analyze the result.

[1] https://github.com/beyondpie/intro_to_stan

Section 1

Introduction

What is STAN [2]

- A *programming language*
  - It supports array data structure, while loops, and conditionals.
  - The syntax much like C++. But no need to worry about C++.
  - STAN itself is written in C++.
  - A model written in STAN needs to be compiled.
- Specifically designed for the *statistical modeling*
  - Vector, matrix and their operations
  - A series of probabilistic functions
  - A series of blocks to describe a statistical model.
  - Support sampling, maximum likelihood estimation and variational inference.

[2] https://mc-stan.org

A typical STAN script: modules

```
## optional
functions {}
## critical module
data {}
## optional
transformed data {}
## critical module
parameters {}
## optioanl
transformed parameters {}
## critical module
model {}
## optional
generated quantities {}
```

Section 2

Behind STAN

MCMC Sampling in STAN

- *Hamiltonian Monte Carlo (HMC)* provides a general sampling procedure for Bayesian inference: using the derivatives of the density function.
- HMC and its adaptive variant the no-U-turn sampler (NUTS) [Hoffman and Gelman, 2014] are used in STAN. NUTS is the default one.
- You DON'T need to write the derivatives in STAN.

HMC in STAN

- Sampling goal: $p(\theta|y)$, the posterior distribution given data y.
- HMC introduces auxiliary momentum variables ρ. $p(\rho, \theta) = p(\rho|\theta)p(\theta)$. In STAN, $p(\rho|\theta) \sim \mathcal{N}(0, M)$, is independent of θ
- Parameters in HMC [See Chapter 15 in STAN Reference Manual]
  - The discretization time ϵ [3] will be automatically optimized during warmup to match an acceptance rate parameter δ (default is 0.8). Increasing δ will force the sampler to use small step sizes, and increase the effective sample size per iteration.
  - The M^{-1} (default a diagonal matrix) is estimated during warmup[4].
  - Number of steps taken L is dynamically adapted during sampling (and during warmup) in NUTS, which is controlled by a predefined parameter *treedepth*.

[3] STAN also allows step-size jitter, which means "jittered" randomly during sampling to avoid poor interactions. Default is 0, producing no jitter.

[4] STAN supports *Euclidean HMC*. M could be configured by the user.

Bounded parameters transformation

Besides auto-tuning the parameters in HMC (NUTS), STAN will transform the bounded variables to an unconstrained ones in the backend.

The basis idea is to set a one-to-one transformation $y = f(x)$:

$$p_Y(y) = p_X(f^{-1}(y))|detJ_{f^{-1}}(y)|$$

Transformations: examples

- $y = \log(x - a)$ if x has lower bound a.
- $y = logit(x) = \log \frac{x}{1-x}$ if $x \in (0, 1)$.
- $y = logit(\frac{x-a}{b-a})$ if $x \in (a, b)$.
- If x is ordered, then $y_k = \log(x_k - x_{k-1})$.
- If $x_k > 0, \sum_k^K x_k = 1$, then
  $y_k = logit(z_k) - log(\frac{1}{K-k}); z_k = \frac{x_k}{1-\sum_{s=1}^{k-1} x_s}, z \in \mathcal{R}^{K-1}$.
- When x is a correlation matrix, find the upper triangular w such that $x = ww^T$, which can be done Cholesky decomposion. Then an inverted transformation is designed on w [See chapter 10 in STAN Referenc Manual for details].

STAN Math Library

- STAN owns a mathematical library [Carpenter et al., 2015] named STAN Math Library that can automatically get the gradients (not the numerical approximation) and support matrix operations ,linear algebra, most common probability functions and so on.
- It's a C++, reverse-mode automatic differentiation library. Not limited to STAN, and designed to be extensive, efficient, scalable and so on.

Reverse-Mode Automatic Differentiation: an example

$$f(y,\mu,\sigma) = -\tfrac{1}{2}(\tfrac{y-\mu}{\sigma})^2 - \log\sigma - \tfrac{1}{2}\log 2\pi$$

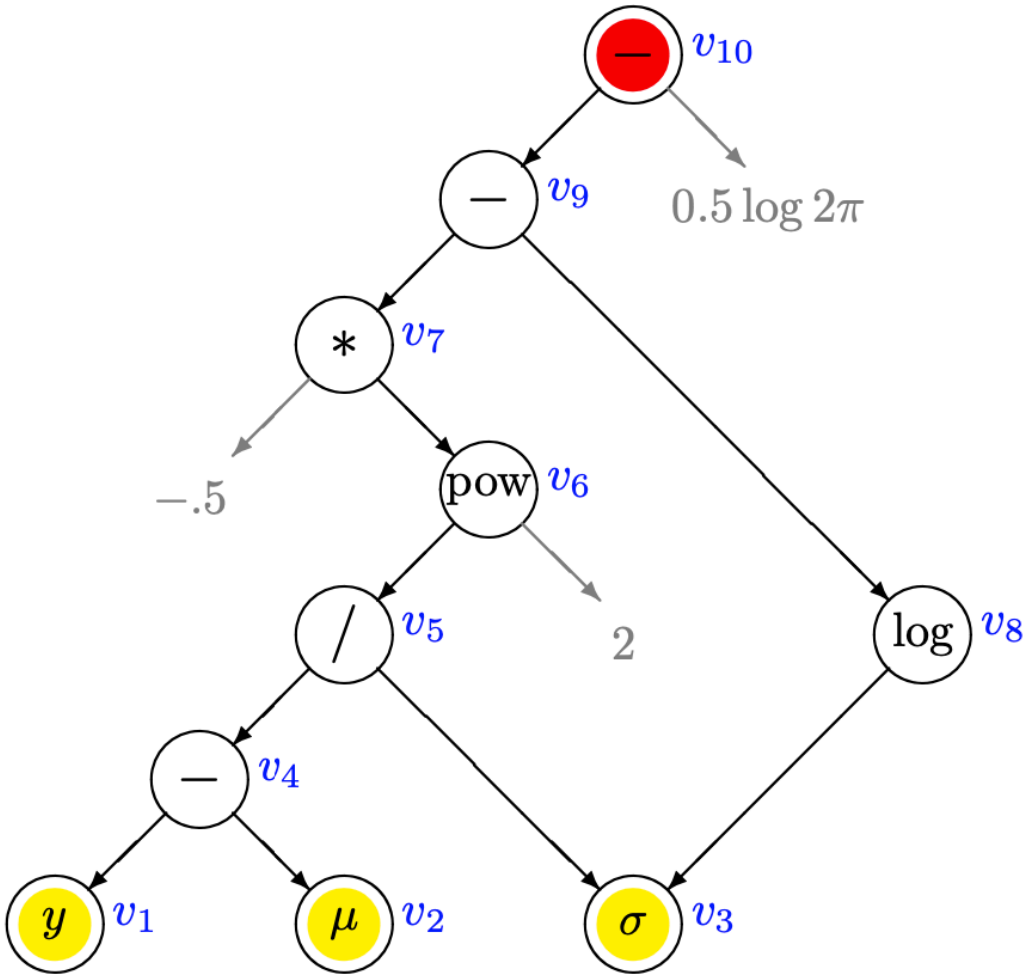


Figure 1: Expression graph of the normal log densityfunction (carpenter2015stan)

Section 3

RUN STAN

How to use STAN with R?

STAN development team provides lots of choices for R users.

- **RStan**: the R interface to STAN.
  - One can fit the model in R and access the output. It relies on Rcpp to call C++ code, and hard to keep updates with the newest STAN.
- **CmdStan**: command-line / shell interface.
  - One can control the compiling easily, e.g., add multi-thread support and GPU support. It directly uses the latest STAN, but not flexible.
- **CmdStanR**: a lightweight interface to CmdStan. You can control CmdStan without leaving R.
- **brms**[Bürkner, 2016]: will make the STAN script for you and pass it to **RStan**.
- rstanarm
  - Bayesian applied regression modeling (arm) via rstan. You can use the customary R modeling sytax, like *glm* with a *formula* and *data.frame*.

Let's consider a simple example.

- Suppose we have N binary observations $y_1, y_2, \ldots, y_N$. They are the *i.i.d* samples from a $Bernoulli$ distribution under the parameter θ.
- Our goal is to infer θ.

Set up the STAN env in R.

```
library(cmdstanr)
# Note: the cmdstan home path is from my computer.
set_cmdstan_path(path = paste(Sys.getenv("HOME"),
                 "softwares",
                 "cmdstan-2.23.0", sep = "/"))
library(bayesplot)
library(posterior)

cmdstan_path()
[1] "/Users/beyondpie/softwares/cmdstan-2.23.0"
cmdstan_version()
[1] "2.23.0"
```

STAN script

```
bern_mod <- cmdstan_model("bernoulli.stan",
                          ## STAN need to be complied.
                          compile = TRUE)
## show the content in the stan script.
bern_mod$print()
data {
  int<lower=0> N;
  int<lower=0,upper=1> y[N];
}
parameters {
  real<lower=0,upper=1> theta;
}
model {
    // uniform prior on interval 0, 1
  theta ~ beta(1,1);
  y ~ bernoulli(theta);
}
```

Let's feed it some data

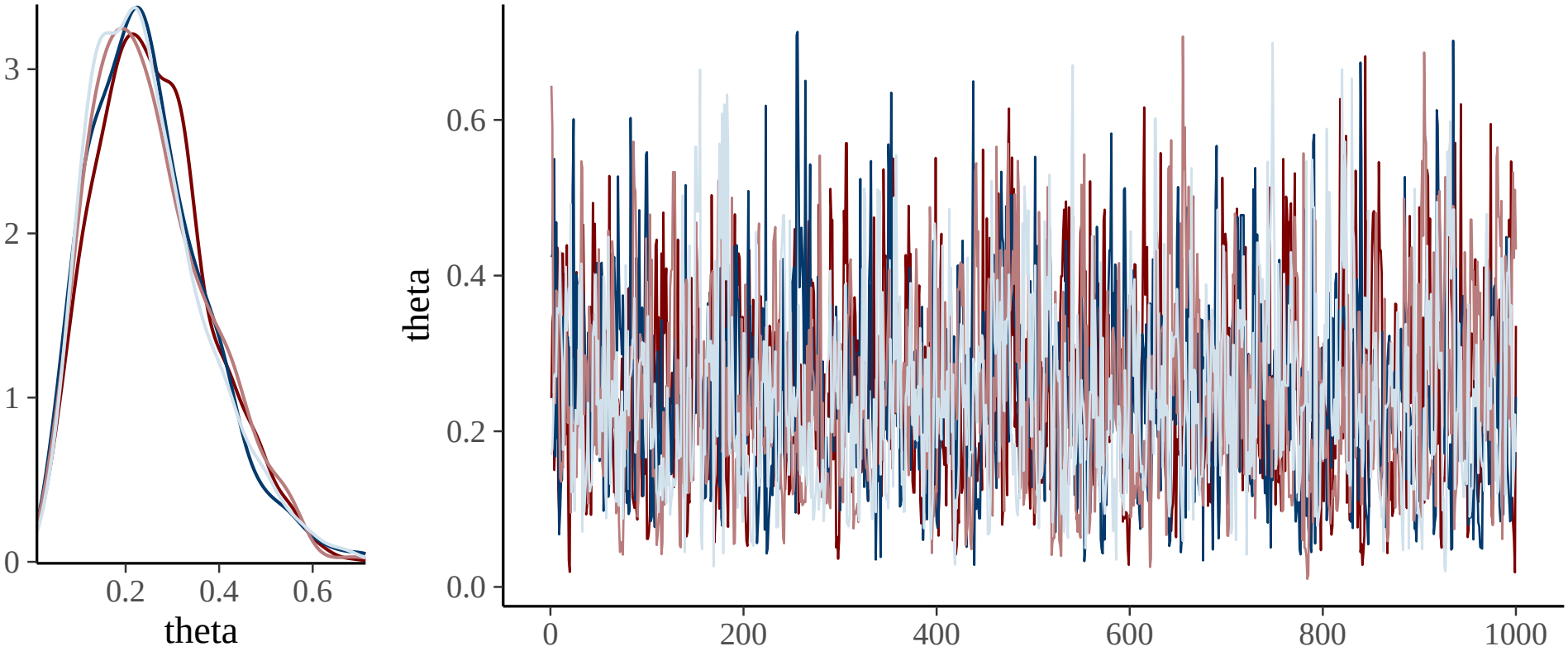
```
bern_data  <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))
## Run MCMC using the 'sample' method
bern_mcmc <- bern_mod$sample(data = bern_data,
                             seed = 355113,
                             chains = 4,
                             parallel_chains = 2,
                             show_message = FALSE)
```

Posterior draws

```
draws <- bern_mcmc$draws()
str(draws)
 'draws_array' num [1:1000, 1:4, 1:2] -6.75 -6.97 -7.04 -7.15 -6.87
 - attr(*, "dimnames")=List of 3
  ..$ iteration: chr [1:1000] "1" "2" "3" "4" ...
  ..$ chain    : chr [1:4] "1" "2" "3" "4"
  ..$ variable : chr [1:2] "lp__" "theta"
## use posterior::as_draws_df to transorm the results
## into data.frame.
```

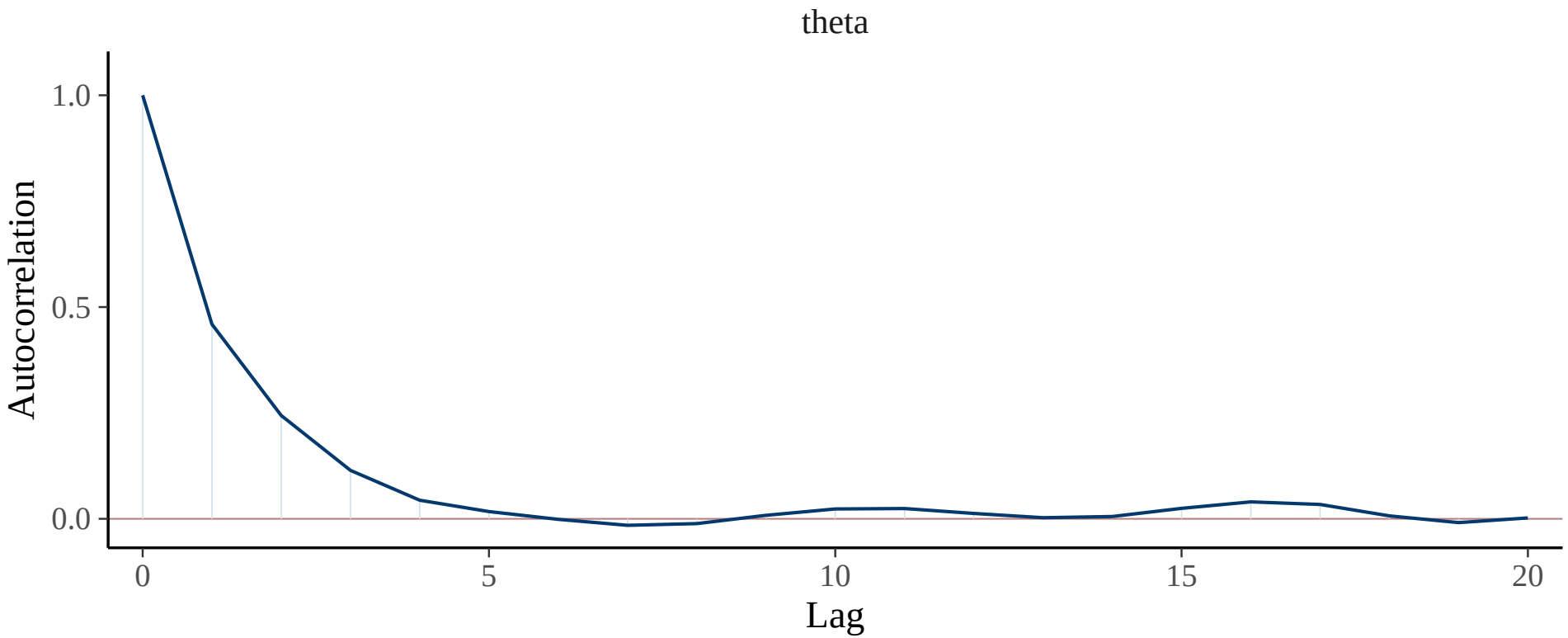
Posterior draws

```
color_scheme_set("mix-blue-red")
bayesplot::mcmc_combo(draws,
                      pars = c("theta"), widths = c(1,3),
                      combo = c("dens_overlay", "trace"),
                       gg_theme = legend_none())
```



Posterior draws

```
bayesplot::mcmc_acf(posterior::as_draws_df(draws),
                    pars = "theta")
```



Posterior Analysis

- $\hat{R}$ statistic
  - An MCMC sample consists of a set of sequence of M (default is 4 in STAN) Markov samples, each consisting of an ordered sequence of N draws from the posterior.
  - The $\hat{R}$ [Gelman et al., 1992] statistic measures the ratio of the average variance of samples within each chain to the variance of the pooled samples across chains. Each chain is split into two halves before STAN calculates $\hat{R}$.
  - If all chains are at equilibrium, $\hat{R}$ will be one. In practice, STAN suggests $\hat{R}$ should be below 1.1.
- Effective sample size(ESS)
  - N_{eff} is defined in terms of the autocorrelations within the sequence at different lags, $N_{eff} = \frac{N}{1+2\sum_t \rho_t}$.
  - STAN estimates N_{eff} with the autocorrelations (using FFT[Geyer, 2011]) and $\hat{R}$ based on multiple chains.
- MCMC Standard Error: $\hat{\sigma_n}/\sqrt{N_{eff}}$.

STAN diagnosis I

```
## use `posterior` package
bern_mcmc$cmdstan_diagnose()

## Checking sampler transitions treedepth.
## Treedepth satisfactory for all transitions.

## Checking sampler transitions for divergences.
## No divergent transitions found.

## Checking E-BFMI - sampler transitions HMC potential energy.
## E-BFMI satisfactory for all transitions.

## Effective sample size satisfactory.

## Split R-hat values satisfactory all parameters.
## Processing complete, no problems detected.
```

Summary of the sampling in STAN

```
## use `posterior` package
bern_mcmc$summary("theta", "mean", "sd" , "rhat",
                  "ess_bulk")
```

variable	mean	sd	rhat	ess_bulk
theta	0.2523255	0.1232632	1.001693	1425.133

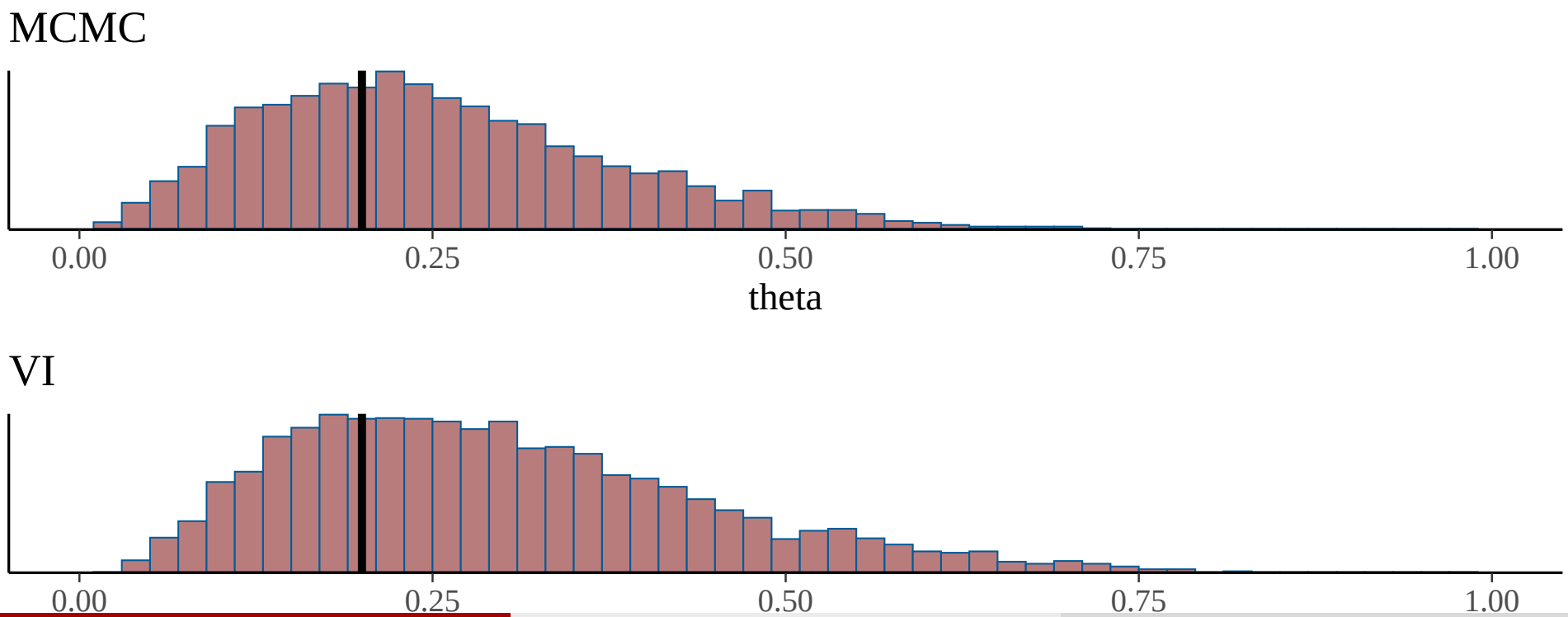
How about variational inference and MLE estimation

```
bern_vb <- bern_mod$variational(data = bern_data,
                                seed = 355113,
                                output_samples = 4000)
```

```
bern_mle <- bern_mod$optimize(data = bern_data,
                              seed = 355113)
```

Result Summary

```
bayesplot_grid(
  mcmc_hist(bern_mcmc$draws("theta"), binwidth = 0.02) +
  vline_at(bern_mle$mle(), size = 1.2),
  mcmc_hist(bern_vb$draws("theta"), binwidth = 0.02) +
  vline_at(bern_mle$mle(), size = 1.2),
  titles = c("MCMC", "VI"),
  xlim = c(0,1))
```



Section 4

Materials and Summary

STAN Materials

- STAN Functions
  - Use this as the reference materials. When you want some functions, just search it.
- STAN Reference Manual
  - You can scan this if you want to know the whole picture of STAN syntax.
- STAN User Guide
  - After the introduction, you could read this document smoothly.
  - Lots of examples cover different statistical modelings.
  - It could be a good material to learn statistical models.
- STAN discourse
  - Friendly and active!

Summary

- STAN is designed as a statistical programming language.
  - Rich of elements, such as matrix operations, probabilistic functions.
  - The clear structures of the modules/blocks for describing a model.
  - Efficiency: C++ backend, multi-thread support, GPU support and map-reduce support.
- STAN provides a general and solid inference framework.
  - Uses *NUTS* and *HMC* for sampling.
  - Uses *ADVI*[Kucukelbir et al., 2017] for variational inference.
  - Use *L-BFGS* for optimization, such as MLE estimation.
- STAN has lots of APIs for R.

Discussion

- STAN does not support sampling discrete parameters. The Rao-Blackwellized estimators must be used: marginalizing out the discrete parameters [5].
- Modern neural network tools, like tensorflow, pytorch: they also support the automatic differentiation, and what's more, they love GPU and eating different kinds of data. So recently, pyro and other tools are developed for "deep" probabilistic programming.

[5] https://statmodeling.stat.columbia.edu/2020/01/29/rao-blackwellization-and-discrete-parameters-in-stan/

Thanks!

- You can find this presentation at https://github.com/beyondpie/intro_to_stan.
- Any suggestions or Pull Requests are welcome.

Paul-Christian Bürkner. brms: An r package for bayesian generalized linear mixed models using stan. *J Stat Softw*, 2016.

Bob Carpenter, Matthew D Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. The stan math library: Reverse-mode automatic differentiation in c++. *arXiv preprint arXiv:1509.07164*, 2015.

Andrew Gelman, Donald B Rubin, et al. Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472, 1992.

Charles J. Geyer. Introduction to Markov chain Monte Carlo. In Steve Brooks, Andrew Gelman, Galin L. Jones, and Xiao-Li Meng, editors, *Handbook of Markov Chain Monte Carlo*, pages 3–48. Chapman and Hall/CRC, 2011.

Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1): 1593–1623, 2014.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.